Übung 10: PHP OOP Graph-Traversierung

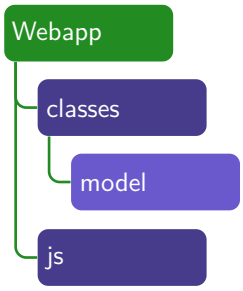
Abgabedatum: 09 Jan. 2020 , 12:00 Uhr

Aufgabe 1: PHP OOP - Graphen

2+2+2+3+2=11 Punkte

Als Vorarbeit für unsere Webanwendung erstellen wir in dieser Aufgabe das Grundgerüst für die Modellierung des Netzgraphen bestehend aus Haltestellen und Linienverbindungen zwischen diesen.

Betrachten Sie zunächst die folgende Ordnerstruktur, welche wir für das finale Projekt anstreben:



- Dateien im Wurzelverzeichnis sollen direkt vom Webserver ausgeliefert werden
- Im Verzeichnis `classes` werden PHP Dateien angelegt, welche auf dem Webserver ausgeführt werden
- Im Verzeichnis `classes/model` werden PHP Klassen angelegt, welche im wesentlichen Datenklassen ohne Programmlogik implementieren
- Im Verzeichnis `js` wird JavaScript Code angelegt

Erstellen Sie die angegebene Ordnerstruktur und behalten Sie diese auch für alle künftigen Aufgaben zur Webanwendung bei.

Für diese Aufgabe werden wir im Verzeichnis `classes/model` Klassen für einen Netz-Graphen anlegen. Dazu modellieren wir Buslinien, Haltestellen und Verbindungen sowie Abfahrten zwischen Haltestellen entsprechend des folgenden Klassendiagramms:

Node
-nodeID: int -edges: List<Edge>
+addEdge(edge: Edge) +getEdge(endNode: Node): Edge

Edge
-endNode: Node -cost: int -line: Line

Line
-id: int -display: String -heading: int

Graph
-nodes: List<Node>
+addNode(id: int) +addEdge(startId: int, endId: int, cost: int, line: Line) +findNode(id: int): Node +print()

Departure
-line: int -display: String -time: DateTime

PathNode
-id: int -cost: int -line: Line

Nicht angegeben sind getter-Methoden (alle Attribute der Klassen sind als **private** zu verstehen), welche aber trotzdem zu implementieren sind. Ebenfalls ist für alle Klassen ein sinnvoller Konstruktor zu implementieren.

a) Implementieren Sie die `Node` Klasse, welche Haltestellen modelliert. Die Methode `addEdge` fügt eine Kante zur Liste der ausgehenden Kanten hinzu, welche in `edges` gespeichert ist. Die Methode `getEdge` gibt die Kante aus `edges` zurück, die zu dem übergeben Knoten führt bzw. `null`, wenn keine Kante zu dem übergeben Knoten existiert.

b) Implementieren Sie die `Line` Klasse. Im Feld `heading` wird die ID der Endhaltestelle gespeichert, die die Linie anfährt. Der String in `display` gibt an, welchen Namen die Linie anzeigt. *Die bekannten Liniennummern (Linie 2, 5, ...) sind nicht äquivalent zu den ids der Linien!*

c) Implementieren Sie die `Edge` und `Departure` Klasse entsprechend des Diagramms (das Feld `line` der `Departure` Klasse enthält die id der abfahrenden Linie).

d) Implementieren Sie die `Graph` Klasse. Die `addNode` und `addEdge` Methoden sollen, falls noch unbekannte IDs für Haltestellen (Parameter `id` bzw. `startId` und `endId`) übergeben werden, neue Knoten erzeugen. Die Methode `findNode(id)` liefert, falls vorhanden, die `Node` Instanz im Graphen, die die gegebene ID besitzt. Die Methode `print` soll für jeden Knoten im Graphen eine Zeile mit der id des Knotens gefolgt vom String `->` und den ids aller Knoten, die mit dem Knoten über eine Kante verbunden sind ausgeben (getrennt durch Leerzeichen).

Aufgabe 2: Graph Traversierung

4+2=6 Punkte

Auf späteren Aufgabenblättern werden wir die Klassen mit Daten unseres Webservers instantiieren und eine sinnvolle Routensuche implementieren. Für den Anfang reicht es uns aber aus, eine beliebige Verbindung zwischen zwei Haltestellen zu finden.

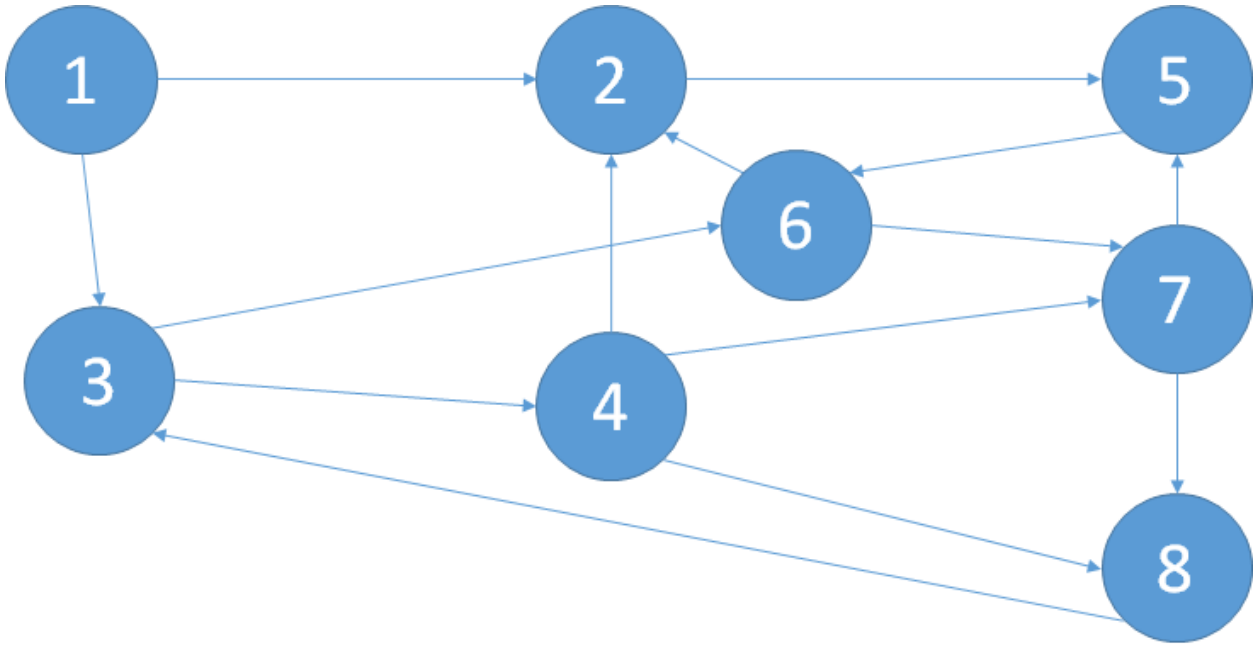
a) Legen Sie im `model` Verzeichnis eine neue Klasse `PathNode` mit den (privaten) Attributen `id`, `cost` und `line` an. Diese modelliert einen Schritt in einem Pfad zwischen zwei Haltestellen:

- `id` – Referenziert die aktuelle Haltestelle
- `cost` – Gibt die "Kosten" (in unserem Fall: die Zeit) an, welche zum Übergang auf die nächste Haltestelle anfallen
- `line` – Referenziert die Linie, welche zur nächsten Haltestelle fährt

Ein Pfad von A nach B kann also durch eine Liste von `PathNode` Objekten beschrieben werden.

b) Legen Sie eine Datei `DFSearch.php` im Verzeichnis `classes` an und implementieren Sie eine Tiefensuche über eine Funktion `dfsearch(graph, startId, endId)`, welche im gegebenen Graphen einen Pfad aus Knoten von der Haltestelle mit id `startId` zu jener mit id `endId` findet und ein Array von `PathNode` Objekten zurückgibt, welche diesen Pfad beschreibt. Für die Rückgabe als Array aus `PathNode` Objekten, schreiben Sie am Besten eine Hilfsmethode, die aus dem Pfad aus `Node` Objekten die entsprechenden `PathNode` Objekte erzeugt. Das erste Element im zurückgegebenen Array ist also ein `PathNode`, welcher auf die Starthaltestelle verweist. Hinweis: Achten Sie auf Zyklen im Graphen. Sie können eine rekursive Funktion implementieren, welche sich als `dfsearchRec(graph, startNode, endNode, visited)` beschreiben lässt.

c) Legen Sie eine Datei `testDFSearch.php` im Wurzelverzeichnis an und erzeugen Sie mithilfe Ihrer Klassen den folgenden Graphen:



Verwenden Sie als Linie und Kosten jeweils den Wert 1. Geben Sie den Graphen mittels `print` Funktion aus. Rufen Sie Ihre Tiefensuche einmal so auf, dass ein Ergebnis gefunden wird und einmal so, dass keines existiert. Geben Sie den gefundenen Pfad bzw. eine Meldung dass kein Pfad existiert geeignet im Browser aus.